An initial approach to distributed adaptive fault-handling in networked systems

SICS Technical Report T2009:07
ISSN 1100-3154

April 2009

Rebecca Steinert and Daniel Gillblad

Swedish Institute of Computer Science,
Box 1263, SE-164 29 Kista, Sweden
{rebste, dgi}@sics.se

Abstract. We present a distributed adaptive fault-handling algorithm applied in networked systems. The probabilistic approach that we use makes the proposed method capable of adaptively detect and localize network faults by the use of simple end-to-end test transactions. Our method operates in a fully distributed manner, such that each network element detects faults using locally extracted information as input. This allows for a fast autonomous adaption to local network conditions in real-time, with significantly reduced need for manual configuration of algorithm parameters. Initial results from a small synthetically generated network indicate that satisfactory algorithm performance can be achieved, with respect to the number of detected and localized faults, detection time and false alarm rate.

Keywords: Adaptive probing, distributed fault-handling, anomaly detection, fault-localization.

1 Introduction

Effective methods for autonomous fault-handling becomes increasingly important with the growing complexity of networked systems. In addition to wired systems, various types of networks have emerged during the last decades, such as wireless networks, sensor networks, mobile networks and ad-hoc networks with dynamic setup. The applications and demands of networked systems have gradually changed and in effect the need for self-managing mechanisms for facilitating fault-handling has increased, for example in networks with varying topology in which nodes are added, removed or replaced and in networks with varying configuration demands etc. In order to maintain quality of services and critical network functionality in such dynamic network environments, autonomous methods for fault-handling that are self-configuring and operative based on local network conditions are necessary in order to e.g. reduce configuration demands,

shorten the reaction time of detected faults and increase the overall efficiency of fault-handling. To qualify for these specific requirements, methods need to be adaptive to varying conditions, while facilitating analysis of abnormal behaviour, and preferably operate in a distributed manner.

Previous work indicate that anomaly detection and fault-localization are often based on network traffic analysis, such as traffic profiling, signature matching, signal analysis, statistical analysis, etc (e.g. [1, 6, 8–10, 14]). Apart from performing analyzes of network traffic, active probing is an alternative approach to anomaly detection and fault-localization. For example, Rish et al propose a method that uses probe selection in order to detect and localize network faults with reduced traffic overhead [14]. Other methods for probe selection are presented by Natu and Sethi, based on so called probe stations which monitor a set of other nodes on a path [11].

Apparently, many of these methods are developed for centrally managed networks with fixed network topologies and equipment, possibly relying on designated nodes for monitoring. However, in dynamic ad-hoc networks with varying topology and network equipment, distributed methods for autonomous fault-handling are more convenient.

In this report, we present a statistical, distributed approach to adaptive fault detection and localization. In our method we use simple end-to-end transactions between nodes and their neighbors while measuring the reply latencies and the packet drop rate on the links. These measurements are then used as input to our anomaly detection model in order to statistically and locally detect traffic anomalies. For fault-localization of a detected anomaly, nodes collaborate locally in order to resolve and report the detected fault. The method of local link monitoring that we use is somewhat similar to that used in some routing protocols such as BGP (Border Gate Protocol) - however, the purpose differs in that we want to resolve errors to link and node level and report these as efficiently as possible for all entities in the network, while in real-time *adapting* to a wide variety of link qualities, including e.g. very lossy wireless channels, such that the need of manual intervention is significantly reduced. Our fault-localization approach is also similar to heartbeat monitoring for fault-localization in distributed systems (e.g. [15]) or local testing for fault detection (e.g. [13]), but the main difference compared to our method is that these approaches do not resolve link or node failures nor do they adapt fully to continuously varying local network conditions.

The statistical model that we use makes our method to fault-handling relatively insensitive to link quality variations, which allows for a robust anomaly detection with reduced false positives. Since both probing intervals and fault detection mechanisms adapt to local measurements, our approach also reduces link load caused by probe traffic, compared to other methods based on frequent probing with fixed intervals.

2 Adaptive probing and distributed fault detection and localization

Our method for distributed fault-handling is a two-step approach. In the first step, abnormal network behaviour is statistically detected using local measurements at each node, aggregate measures from subnets, and probes, in order to test the accessibility on network level of links and nodes, or the availability of services on the application level. The second step is initialized whenever a node has detected a possibly anomalous node in the first phase, in order to localize the fault to a certain node, process or link.

For the developed algorithm that we will describe, we assume that each node can perform simple end-to-end test transactions (such as ICMP requests) on its neighbours, and measure the latency of this request. We then estimate the parameters of the distribution formed by the local measurements, and use them in our statistical model. Further, each node needs to know the local topology, which in this case means awareness of all other nodes within the topological distance of two. In our simulation framework all nodes are initially aware of the local topology, and thus we have not implemented any particular distributed topology discovery mechanisms. However, in real-world network applications, a relatively easy approach to distributed topology discovery could be that new nodes announces its presence to all its connections in order to receive information about all nodes within two hops, while existing nodes (to which the new node is connected) notify all neighboring nodes such that all neighbour lists are kept consistent.

2.1 Using local probes for detection of abnormal behaviour

For the purpose of both anomaly detection and fault-localization we use adaptive probing, which refers to the adaptation of the total probability distribution of unsuccessful probes using locally extracted information from nodes in the network. In order to detect anomalies so called probing tests are performed, during which a series of probes are sent in order to increase the certainty of a potential anomaly such that the number of false positives are reduced.

We assume that we have the total probability distribution that a probe fails. Unless a response has been received for a probe, the probability of receiving a response is adapted before each new probe. When the probability of a response reaches below a threshold ψ, the network component is considered faulty and the process of fault-localization is triggered.

During run-time, observations of probe reply delays are continuously collected, forming a distribution from which we can calculate the probability of receiving a probe response within a certain time delay. Here, the probability of receiving a probe response within time Δt is

$$P(R_{\Delta t}) = (1 - P(D)) \int_0^{\Delta t} P(t)dt \tag{1}$$

where $P(t)$ is the probability density of probe responses at Δt. Further, we assume that the total probability of a response R given a set of statistically independent probes in a probe test $\{R^{(1)}_{\Delta t}, R^{(2)}_{\Delta t}, \ldots, R^{(n)}_{\Delta t}\}$ is

$$P(\neg R|\Delta t^{(1)}, \Delta t^{(2)}, \ldots, \Delta t^{(n)}) = \prod_{i}^{n}(1 - P(R^{(i)}_{\Delta t})) \tag{2}$$

such that with each failed probe the probability of a response decreases. When the probability of not receiving a response given previous probes has reached below a predefined threshold, a fault has been detected, and the process of fault-localization is triggered. However, if a response is received, eq. (2) is reset to $P(\neg R|\Delta t^{(1)}, \Delta t^{(2)}, \ldots, \Delta t^{(n)}) = 1.0$.

The probability density function $P(t)$ can be any type of distribution that matches the characteristics of the data. Here, $P(t)$ is a Gamma distribution, which is an assumption based on a series of measurements performed on real-world Ethernet networks (see section 3.1).

From distribution $P(t)$, we also determine with which interval probes are sent. This means that anomalies are adaptively detected with the use of short series of probes which effectively reduces link loads caused by probe traffic, compared to ordinary heartbeat-monitoring algorithms in which probes are frequently sent without taking e.g. link quality into account.

The algorithm involves two types of intervals controlled by parameters τ and θ, both based on the expected link latency and the cost of sending a probe. Here, parameter τ is

$$\tau = m_\tau f(x) \tag{3}$$

where m_τ is a multiple based on the cost of sending a series of probes, $f(x)$ is the inverted cumulative density function of $\int_0^{\Delta t} P(t)dt$, and x is a fraction that is used to determine the corresponding delay. In a similar fashion the probing interval θ in a probing test is determined by

$$\theta = m_\theta f(x). \tag{4}$$

.

Parameter θ adaptively determines with which interval probes are sent in a probe test, and is significantly smaller than τ. Parameter τ controls the interval with which a probe test is performed (section 2.3). This way link load caused by induced probing traffic is reduced during normal network behaviour, while being somewhat increased when a potential anomaly is about to be detected. The shorter probing interval θ relative τ allows for a faster anomaly detection with increased certainty, such that the number of false positives is reduced.

It is reasonable to assume that probes sent with interval θ represents a much lower cost than probes sent with interval τ; if a link or node is faulty, the traffic load induced by a higher probing rate is of small significance in the presence of abnormal network behaviour and reduced service quality.

2.2 Fault-localization

Collaborative fault-localization is triggered whenever a node has detected an anomaly on the connection to any of its neighboring nodes. The purpose of the fault-localization process is to identify the root cause of the anomaly. Here, we address two types of faults - node failures and link failures. In order to identify whether the anomaly is caused by a node failure or a link failure, the neighbors of the node to which the communication has failed are involved in the fault-localization process.

We assume that each node n has a list of all neighbouring nodes within two hops. The rate at which each node will probe a neighbour is determined locally as described. Whenever a probe test from node n to a node $\hat{n}$ fails, node n will initiate a localization process involving collaboration with the neighbouring nodes of $\hat{n}$, $\{\tilde{n}, \tilde{n}_2, \ldots, \tilde{n}_i\}$, in order to test the connection to $\hat{n}$ and report back to n. If at least one of the nodes $\tilde{n}_i$ reports a successful probe response, it is concluded that the anomaly was caused by a link failure, otherwise the anomaly was caused by a node failure in $\hat{n}$ (fig. 1a). When the root cause has been determined, the fault-localizing node n reports the fault to specified recipients (e.g. the network operations center). If the anomaly was caused by a link failure, information about the fault is spread by involved collaborating nodes to the other node on the faulty link, in order to increase the fault-localization efficiency. If instead the anomaly was caused by a node failure, all neighbors of the faulty node are made aware of the fault.

However, in some cases the root cause of a detected anomaly is undecidable. For example, consider the case that the only link between a probing node and the inaccessible node is down and that there is no other route available for communication between the neighbours of the inaccessible node (fig. 1b). In such a situation it is undecidable whether it is the node or the link that has failed, and the nodes that are affected will notify relevant recipients.

A similar situation might occur if the neighbouring node of the inaccessible node is unavailable such that the probing node cannot request assistance from collaborating nodes in order to confirm the fault (fig. 1c). In that case the node that detected the anomalous behaviour will eventually notify relevant recipients that the root cause of the anomaly is undecidable, because of the failure to communicate with the collaborating node $\tilde{n}$. In our implementation, this notification will be sent after t_r seconds has passed, counted from the beginning of the fault-localization process (section 3.6).

2.3 Formal algorithm description

In this section, we describe the subroutines forming our anomaly detection algorithm. Let n be a node in the network. Each node needs to keep track of the set of neighbouring nodes, N_n, as well as the sets of neighbours to each of these neighbours i, N_n^i. As an example of how the algorithm can be set up when run in networks under e.g. churn, we also provide descriptions of the procedures when nodes are connected or disconnected to the network. The procedures

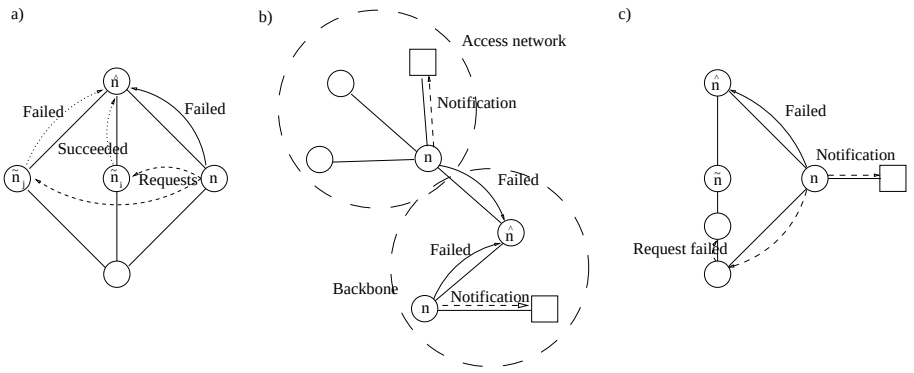


Fig. 1: Figure a) shows the normal case. Node n cannot reach $\hat{n}$ and sends requests to $\tilde{n}_i$,$\tilde{n}_j$ to test the connection of $\hat{n}$. Since the probe between $\tilde{n}_i$ and $\hat{n}$ is successful, the link between n and $\hat{n}$ is considered broken. In figure b) n cannot access $\hat{n}$ or its neighbor. In that case n directly notifies relevant recipients (e.g. a network operations center). In figure c) the neighbouring node $\tilde{n}_i$ is unaccessible due to a faulty intermediate node on the path. In this case the probing node n notifies relevant recipients that the cause of the detected anomaly is undecidable.

taken when connecting node $\hat{n}$ to n in the network are described in subroutines 1 and 2. These procedures have not been implemented in our simulations - instead we have assumed that the network topology is fixed and that all nodes are aware of all nodes within two hops.

Let each node n store an *error state* S_n^i for each neighbour i. Each S_n^i represents the current state of n_i as viewed from n, and can be assigned one of the following values:

- *No fault.* No fault has been detected in the neighbour.
- *Link or node failure.* A fault has been detected, but it has not been or it is not possible to determine if it is a link or node failure.
- *Link failure.* A fault has been detected and diagnosed as a link failure.
- *Node failure.* A fault has been detected and diagnosed as a node failure.

Algorithm 1 Connect node $\hat{n}$ to node n

Require: Valid nodes n and $\hat{n}$
$N_n \leftarrow N_n \cup \{\hat{n}\}$
$N_n^{\hat{n}} \leftarrow N_{\hat{n}} \setminus n$
Monitor node $\hat{n}$ from n

Algorithm 2 Disconnect node n

Require: Valid node n
  for all $\hat{n} \in N_n$ **do**
    $N_{\hat{n}} \leftarrow N_{\hat{n}} \setminus \{n\}$
    for all $\tilde{n} \in N_n^{\hat{n}}$ **do**
      $N_{\tilde{n}}^{\hat{n}} \leftarrow N_{\tilde{n}}^{\hat{n}} \setminus \{n\}$
    end for
  end for

Algorithm 3 Monitor node $\hat{n}$ from node n

Require: $\hat{n} \in N_n$
  repeat
    if Test node $\hat{n}$ from n fails **then**
      if $S_n^{\hat{n}} =$ *No fault* **then**
        for all $\tilde{n} \in N_n^{\hat{n}}$ **do**
          Confirm failure of $\hat{n}$ for n in $\tilde{n}$
        end for
        if Any $\tilde{n} \in N_n^{\hat{n}}$ report success **then**
          $S_n^{\hat{n}} \leftarrow$ *Link failure*
          Report failed link from n to $\hat{n}$
        else if All $\tilde{n} \in N_n^{\hat{n}}$ report failure **then**
          $S_n^{\tilde{n}} \leftarrow$ *Node failure*
          for all $\tilde{n} \in N_n$ **do**
            $S_{\tilde{n}}^{\hat{n}} \leftarrow$ *Node failure*
          end for
          Report failed node $\hat{n}$
        else
          $S_{\tilde{n}}^{\hat{n}} \leftarrow$ *Link or node failure*
          Report link or node failure
        end if
      end if
    else
      if $S_n^{\hat{n}} \neq$ *No fault* **then**
        $S_n^{\hat{n}} \leftarrow$ *No fault*
        if $S_n^{\hat{n}} =$ *Node failure* **then**
          for all $\tilde{n} \in N_n$ **do**
            $S_{\tilde{n}}^{\hat{n}} \leftarrow$ *No fault*
          end for
        end if
        Report working link from n to $\hat{n}$ and node $\hat{n}$
      end if
    end if
    Wait τ s
  until $\hat{n}$ disconnects

Algorithm 4 Test node $\hat{n}$ from n

Require: $\hat{n} \in N_n$
  repeat
    Send test transaction to $\hat{n}$
    Wait θ s
  until Any response or $\prod_i (1 - P(R_{\Delta t}^{(i)})) < \psi$
  if Any response **then**
    return Success
  else
    return Failure
  end if

Algorithm 5 Confirm failure of $\hat{n}$ for n in $\tilde{n}$

Require: $\hat{n} \in N_{\tilde{n}}$, $n \in N_{\tilde{n}}^{\hat{n}}$
  $t \leftarrow$ Test node $\hat{n}$ from $\tilde{n}$
  Report t to n

3 Simulation environment and implementation

We have implemented the algorithm in the discrete event simulator environment OMNET++ [16], in which we simulate link latencies, fault events and varying link qualities, such as drop rates. The OMNET++ was chosen as simulation environment because of the modular framework and graphical user interface, which not only fits our simulation needs, such as generation of traffic, fault events etc, but also allows for future graphical demonstrations of our approach. Further, the modular framework makes it relatively easy to add future extensions to our fault-handling approach.

3.1 Parameter estimation

Based on empirical probe testing and latency measurements on Ethernet links in different types of real-world networks (fig. 2 and 3), we assume that the distribution of link latencies is Gamma distributed. Similar conclusions about network traffic matching Gamma, Weibull, or other exponential distributions have been made in a number of other papers, e.g. [2–5, 7].

During simulation, the Gamma parameters are estimated and used to calculate the probability of probe responses described in (1) and (2). For this purpose we use a simple method of moments approach to estimate the parameters scale α and shape β from the first and second sample moments of the latency observations collected during runtime. The motivation to our choice of estimation approach is mainly to reduce computational demands, although there are more accurate methods for parameter estimations. For example, it would be possible to perform a maximum likelihood estimation using some numerical approximation algorithm (e.g. Newton-Raphsons method), combined with the moment estimation as an initial guess. However, since there is uncertainty in the data, the need

for high-precision estimates is in this case reduced, and hence we have chosen to prioritize computational speed above accuracy.

Further, when a node is connected to the network, it sends a series of probes to quickly obtain an initial estimate of α and β of the latency distributions to each of its neighbors. When the node has obtained k successful probe responses, the main algorithm is started, using the initially estimated parameters.

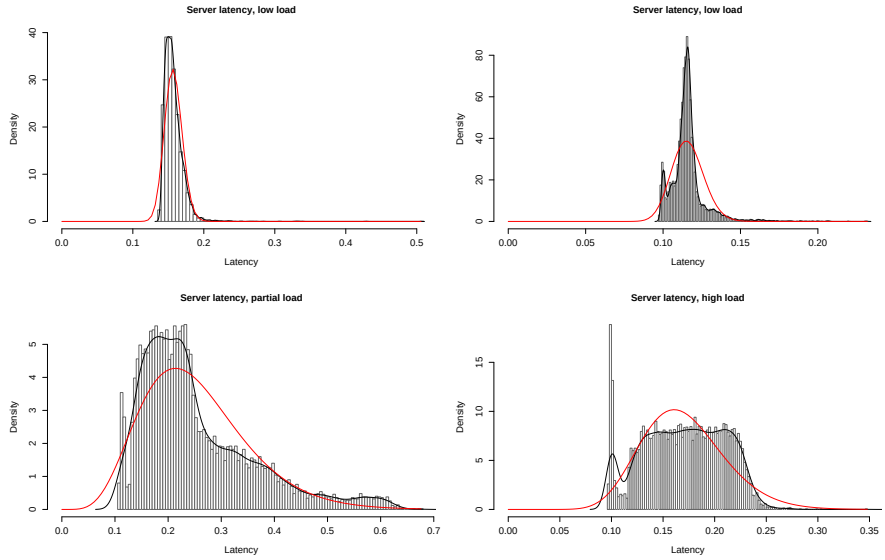


Fig. 2: Examples of latency distributions on a wired network. Starting at the upper left, the graphs describe the latency distribution on a local Ethernet link on servers with no, low, medium, and high load respectively.

3.2 Routing

We have implemented simple routing tables based on the shortest path from one node to another. However, in real-world network applications it would be more convenient to implement alternative paths, in order to increase the collaboration efficiency between nodes involved in the localization processes of detected anomalies. This is relevant in situations where the rate of failure events in a local region of the network is high, for example.

3.3 Link latency

Randomly selected Gamma parameters, drawn from a normal distribution, are used to symmetrically simulate traffic latencies in both directions of each link.

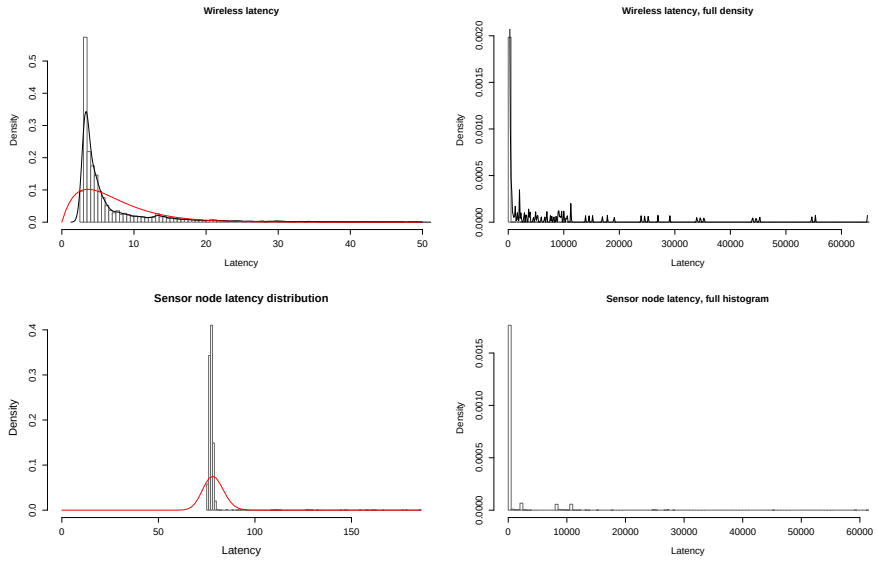


Fig. 3: Examples of latency distributions on wireless networks. The upper two graphs represent latencies in a 802.11b network with very low signal strength, and the lower two the latencies in a very simple sensor node. The graphs to the left show estimated gamma distributions on truncated data, while the graphs to the right show (exaggerated) versions of the actual histograms, indicating that in both applications there are connection problems that will be reported as intermittent error in our algorithm.

During simulation, each data packet is sent with a latency drawn from the Gamma distribution with random parameters. Further, we make no assumptions of the queueing order of data packets since only the reply latencies are of interest to measure. This means that data packets are not always processed in the order of transmission. Since the model is based on probe reply delays, the ordering of packets is in this case irrelevant.

3.4 Fault event generation

Fault events are randomly generated over the whole population of nodes and links, drawn from a Poisson distribution with parameter λ specifying the expected number of fault events within a given time period. The Poisson distribution is a natural choice for modelling fault events occurring in a population of the equipment in distributed systems.

3.5 Traffic drop rate

The amount of dropped packets is adjusted by the drop parameter ξ on a link in one direction, such that a fraction p of randomly chosen packets are dropped

when sent from a node. This means that the effective drop rate that affects a node on an undirected link converges toward $2p = P(D)$ (eq. 1). In the experiments, the drop rate is equal in both directions on all links and randomly drawn from a Gaussian distribution with mean ξ and deviation $\sigma = 0.2\xi$.

3.6 Fault modes and state transitions

Each node operates in two modes with respect to a neighboring node, namely in abnormal or normal mode. A node has also four different states related to the anomaly detection and fault-localization processes, namely `IDLE`, `BUSY_PROBING`, `WAITING_FOR_RESULT` and `WAITING_FOR_NOTIFICATION`.

Normal and abnormal mode In normal mode, all probe tests sent from node n to neighbor $\hat{n}$ are successful and the communication is working free from any local network anomalies. When n detects an anomaly in the communication with $\hat{n}$ and the fault has been localized and confirmed by the neighbors $\tilde{n}$ of $\hat{n}$, node n makes a transition into abnormal mode with respect to $\hat{n}$. In this mode, n continues to send probes to $\hat{n}$ but does not request confirmation from $\tilde{n}$ as long as the probe tests fail. As soon as any reply from $\hat{n}$ is obtained, the node makes a transition back to normal mode again.

IDLE state When node n is in `IDLE` state with respect to $\hat{n}$, it sends probes and operates either in normal or abnormal mode as described above.

BUSY_PROBING state The node n makes a transition from `IDLE` state to `BUSY_PROBING` state if any node $\tilde{n}$ has requested confirmation of a detected anomaly in communication with $\hat{n}$. Node n immediately sends probes to $\hat{n}$ in order to confirm the communication failure. If several nodes request confirmation from n about the same node $\hat{n}$, the outcome of the probe test performed by n will be sent to all requesting nodes.

WAITING_FOR_RESULT state When node n has detected an anomaly in communication with neighbor $\hat{n}$, it makes a transition into the `WAITING_FOR_RESULT` state and requests confirmation from collaborating nodes $\tilde{n}$. For a duration of t_r seconds, n waits for all $\tilde{n}$ to transmit the outcome of the confirmation procedure with respect to $\hat{n}$. If the anomaly can be concluded to be either a link failure or a node failure in collaboration with $\tilde{n}$ within time t_r, node n notifies all relevant recipients (including nodes $\tilde{n}$). Node n then makes a transition into `IDLE` state and starts to operate in abnormal mode with respect to node $\hat{n}$.

In case t_r seconds has passed before the true failure has been localized, n makes a transition to abnormal mode with respect to $\hat{n}$ and notifies relevant recipients about the undecidable fault. The reason to restrict the duration of this state is to prevent the node from waiting infinitely for confirmation from

other nodes. This situation may occur if, for example, the node itself fails, or if the communication fails on the path between n and $\tilde{n}$ (as shown in section 2.2).

It should be pointed out that several nodes can transition into this state at more or less the same time. In some situations (e.g. in network regions with bad link quality), node failures can cause several neighboring nodes to almost simultaneously initialize a fault-localization process to confirm abnormal behaviour of a common node. This means that a node that initiated a fault-localization process can also be part in confirming a detected anomaly in another node. In this case, confirmation information is directly exchanged between collaborating nodes, without executing the fault confirmation (i.e. probe tests) process. In other words, since these nodes have already detected an anomaly, the failure to communicate with $\hat{n}$ is directly confirmed and sent to node n that initiated the fault-localization process. This behaviour is implemented in the simulator to increase the robustness of the fault-localization process for node failures, such that at least one of the initiating nodes n will be able to receive confirmation results from all of its collaborators $\tilde{n}$. We chose to implement this behaviour because of its simplicity, however an alternative solution would be to implement a negotiation mechanism instead, such that the fault-localization process is initiated by only one node.

WAITING_FOR_NOTIFICATION state Node n makes a transition from `BUSY_PROBING` state into `WAITING_FOR_NOTIFICATION` state when the requested confirmation procedure of $\hat{n}$ is completed. For t_n seconds, node n waits for a notification from the requesting node that the failure has been localized. In case no such notification is received, the node will eventually change state into `IDLE` when t_n seconds has passed. Here, t_n is a multiple of the expected link latency and the number of neighbors to the node of which the communication has failed. The reason for this duration is to prevent the node to start a new fault-localization process in collaboration with other nodes.

4 Experiments

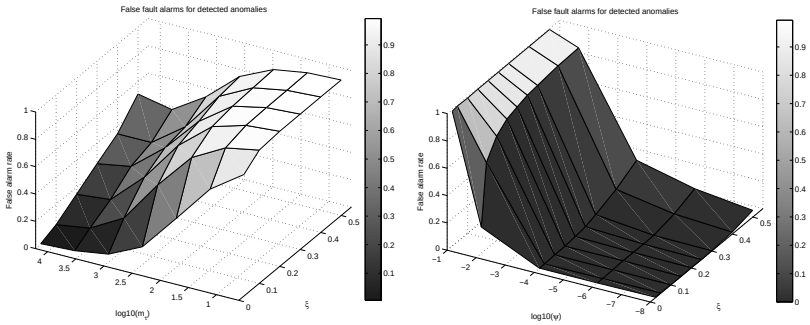
In this section, we investigate different performance aspects of the algorithm. We have tested the algorithm in various simulated conditions. Specifically we have performed a series of experiments in which we have varied the number of fault events and drop rates, while measuring the performance with different parameter settings. For this purpose, the experiments were performed on a synthetically generated network topology. All results shown in this report have been extracted from various log files obtained during the simulations. The objective of our experiments was not to achieve optimal performance results, but to investigate algorithm performance with varying parameter values. In all our experiments, we used a synthetic scale-free network of 30 nodes and 81 undirected links, generated with the Barabsi-Albert method, starting with 5 nodes and $m = 3$ links added at each step [12]. To introduce nodes with single neighbors, 5% of the links were randomly removed. The reason to use a scale-free network is that

the hub-like structure closely resembles the structure of a real-world network topology.

In all the experiments we assumed that in each period of 14400 seconds a mean number of $\lambda = \{10, 20, 40, 60, 80\}$ fault events were generated on randomly selected network equipment with uniform distribution. The fault duration was uniformly random up to 3600 seconds. Simulated link latencies in one direction were based on randomly drawn parameter values from a Gaussian distribution, with $\mu = 0.0025$, $\sigma = 0.0005$ for the scale parameter and $\mu = 30$, $\sigma = 6$ for the shape parameter. Further, we varied the mean drop rate $\xi = \{0.025, 0.1, 0.2, 0.3, 0.4, 0.5\}$. In all of the experiments, we set $t_r = t_n = 64.0\tilde{n}f(0.8)$. The shorter probing interval used in the probe tests were set to $\theta = f(0.8)$. For varying latency thresholds ψ and intervals τ between probe tests, we used different parameter values $\psi = \{10^{-8}, 10^{-6}, 10^{-4}, 10^{-2}, 10^{-1}, \}$ and $\tau = m_\tau f(0.8)$, where $m_\tau = \{4, 16, 64, 256, 1024, 4096, 16384\}$. During initialization, each node sent $k = 200$ probes to obtain preliminary estimates of link latencies. For statistical significance, all results are based on 4 days of simulated time and shown as the mean of 10 runs.

4.1 Reducing false positives by adjusted parameters

In this section, we investigate how the rate of false positives relates to the algorithm parameters in combination with varying rates of drop and network failures.

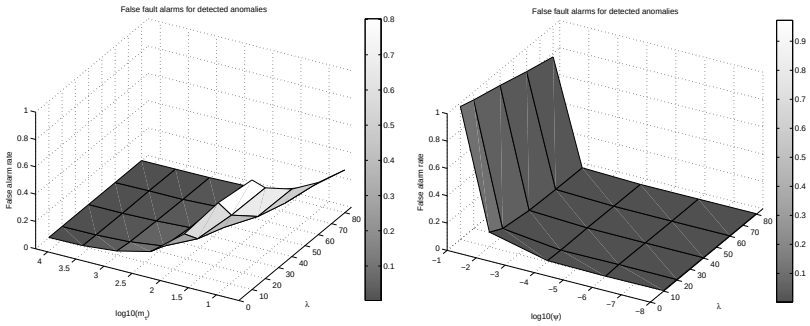


(a) (b)

Fig. 4: False positives of detected anomalies, obtained for varying settings of $\tau = m_\tau f(0.8)$ and $\psi = 0.005$ (fig. 4a) and varying ψ with $m_\tau = 256$ (fig. 4b), in combination with different values of ξ and $\lambda = 5$. The number of false alarms increases for low τ whereas it can be reduced using a very low value on ψ. Parameter settings for optimal performance depends here mainly on the drop rate.

Figure 4 shows the number of false positives relative the number of simulated faults when varying the rate of traffic drops. We observe that when the cost m_τ is low, the algorithm is more sensitive to false alarms than when m_τ is set to a high

cost (fig. 4a). For example, we see that the rate of false alarms produced under high drop rates can be reduced by increasing m_τ. Naturally, with a shorter τ, the detection of anomalies becomes more reactive to very small drop rates since probing tests with interval θ are performed more often. Further, the rate of false positives increases when ψ is set to larger values (fig. 4b). Statistically, a larger ψ value means that the uncertainty demands of detected anomalies are relaxed such that fewer probes are needed in order to detect an anomaly. Consequently, this leads to a higher rate of false positives. In addition, we observe that the number of false positives increases with increasing drop rate and fixed values of ψ (fig. 4b). We also observe that the rate of false positives can be reduced when the drop rate is increased, by adjusting ψ to a smaller value.



(a) (b)

Fig. 5: False positives of detected anomalies, obtained for varying settings of $\tau = m_\tau f(0.8)$ and $\psi = 0.005$ (fig. 5a) and varying ψ with $m_\tau = 256$ (fig. 5b), in combination with increasing failure events λ and drop rate $\xi = 0.025$. We see that the number of false positives can be reduced by adjusting parameter τ and ψ.

For varying number of failure events and fixed drop rate, we observe in fig. 5a that the rate of false positives can be reduced by increasing the cost m_τ. In addition, we see that when the failure rate increases (specifically for small values on m_τ, the false positive rate is reduced since the number of induced faults is higher relative the fixed drop rate. Further, we observe in fig. 5b that the rate of false positives depends on the value of ψ rather than on the number of failure events, compared to the previous case with varying drop rate shown in fig. 4. This indicates that ψ in the future could be set adaptively based on the drop rate (and possibly some other measure) for the purpose of further reducing efforts on manual configuration, instead of having it set equally for all nodes which currently is the case.

4.2 Probing and detection performance

In this section we investigate how the probing interval τ and latency threshold ψ relates to the detection rate of generated fault events, the mean shortest detection time and the number of probes sent to detect an anomaly, while the drop rate ξ and the number of fault events λ are varied. The mean shortest detection time was measured in seconds from the start of a generated fault event to the time of detection and initialization of collaborative fault-localization.

In general, we see in fig. 6 that the detection rate of fault events while varying drop rate ξ, is over 95% for both link and node failures. We observe that the detection rate of generated fault events decreases slowly for small changes in m_τ, up to a certain point where m_τ is set to very large values such that the detection rate drops significantly (fig. 6a, 6b). Apart from some overlapping link and node failures (involving the same nodes and links), other intermittent faults remain undetected as a result of short fault durations less than τ. This applies specifically to links (fig. 6b) rather than to nodes, since one link is monitored only by two nodes whereas nodes are monitored by several other nodes (fig. 6a). For various values on ψ, we see that the detection rates of generated fault events remain quite fixed close to 100% for both node and link failures (fig. 6c, 6d).

In figure 7, we see that when the failure rate varies, more than 95% of the node failures and more than 80% of the link failures can be detected for various values of m_τ (fig. 7a, 7b) and ψ (fig. 7c, 7d). Further, we observe that the detection rate in general is more sensitive to increasing number of fault events, compared to when only the drop rate is varied. With increasing number of fault events, overlapping link and node failures involving the same network elements occur to a higher degree, which in this case explains the increasing number of undetected link failures (fig. 7b, 7d). In addition, some faults also remain undetected due to short fault durations. Finally, we observe that the detection of node failures is in general stable for different values on m_τ and ψ for increasing number of fault events λ, since most nodes are monitored by several other nodes (fig. 7a, 7c).

In figures 8 and 9, we see that faults can be detected with short detection time, less than 10 seconds depending on the parameter settings. The mean shortest detection time, measured from the start of a fault event to the initialization of the fault-localization process, depends primarily on τ rather than on the drop rate and the number of fault events (fig. 8a, 8b and fig. 9a, 9b). Up to a certain point, we see that small values of τ have no significant effect on the detection time. This means that small changes in the interval τ with which probing tests are performed are insignificant relative the detection time, and as such satisfactory results can be achieved with reduced need for fine-tuning τ. Compared to the number of false positives (fig. 4a, 5a) and the detection rates (fig. 6a, 6b and fig. 7a, 7b), we also see that satisfactory performance can be achieved using quite sparse intervals up to a certain point where the performance with respect to varying τ decreases. Thus, our results indicate that we can detect anomalies without generating significantly large amounts of extra traffic load caused by excessive probe testing in our anomaly detection method. Further, we see in

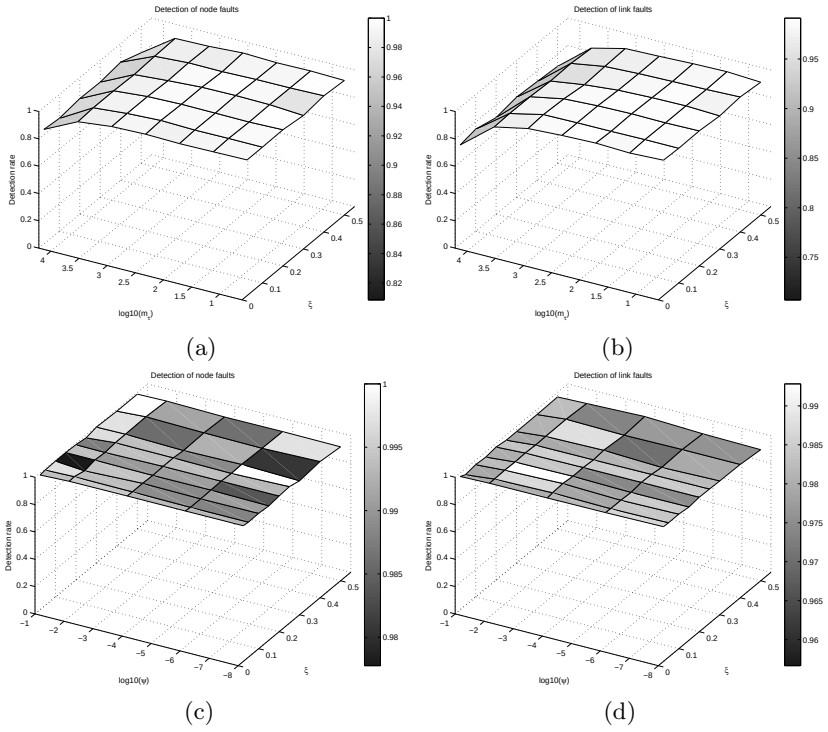


Fig. 6: Detection rates of faults with increasing ξ and fixed $\lambda = 5$. In fig. 6a and 6b, we see results obtained with varying $\tau = m_\tau f(0.8)$ and $\psi = 0.005$, whereas fig. 6c and 6d show experimental results obtained with varying ψ and $m_\tau = 256$. We observe that nearly 100% of the simulated faults can be detected.

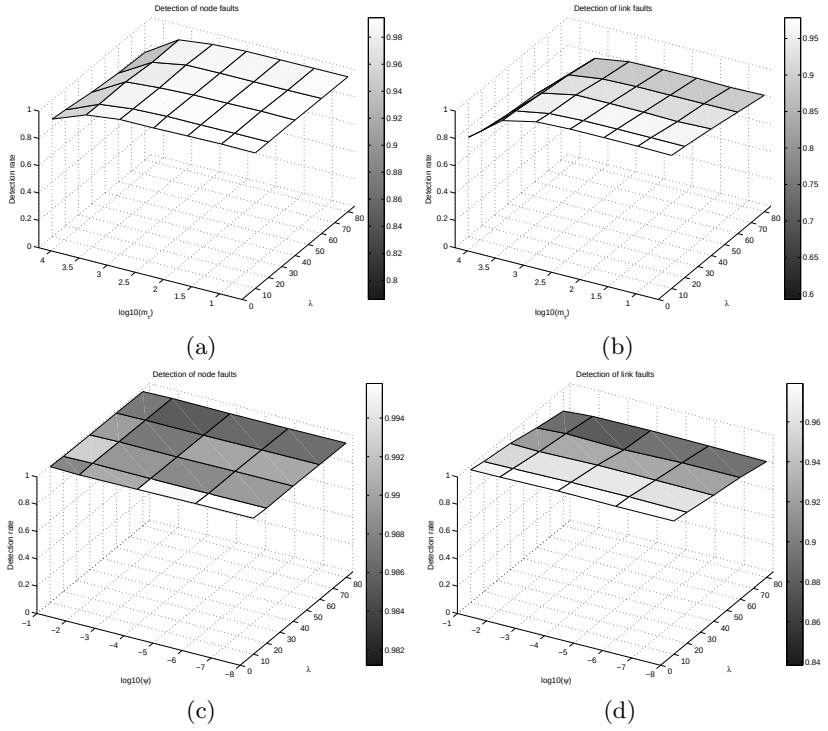


Fig. 7: Detection rates of faults with increasing λ and fixed $\xi = 0.025$. Figures 7a and 7b show results obtained with varying $\tau = m_\tau f(0.8)$ and $\psi = 0.005$, whereas fig. 7c and 7d show detection rates with varying ψ and $m_\tau = 256$. The detection rates vary primarily with the number of fault events and nearly independently from varying parameter settings.

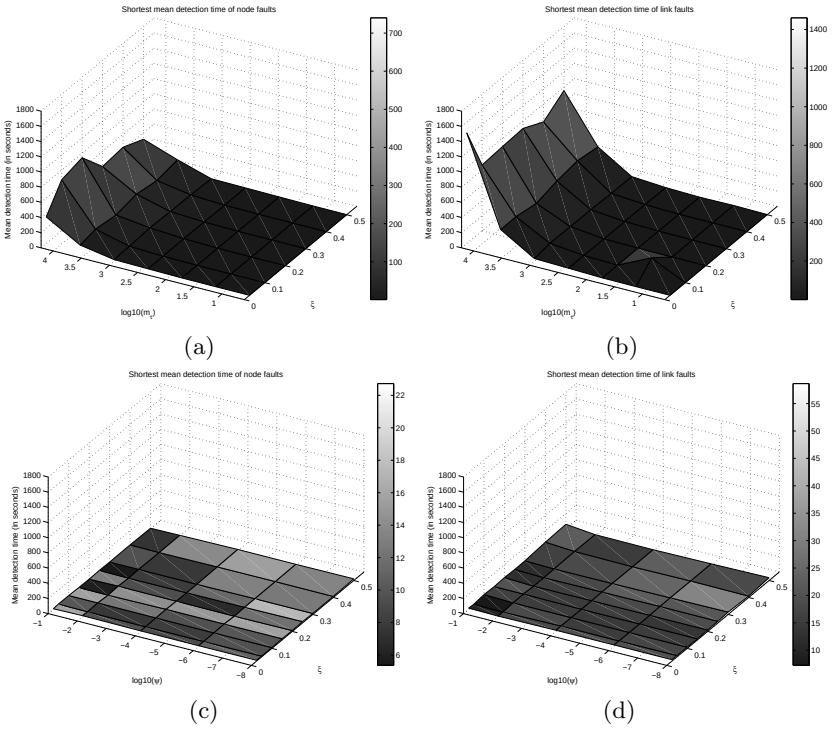


Fig. 8: Detection time with varied ξ and $\lambda = 5$. In fig. 8a-8b, we see results obtained with varying $\tau = m_\tau f(0.8)$ and $\psi = 0.005$, whereas fig. 8c-8d show results with varying ψ and $m_\tau = 256$. The detection time can be controlled by adjusting τ.

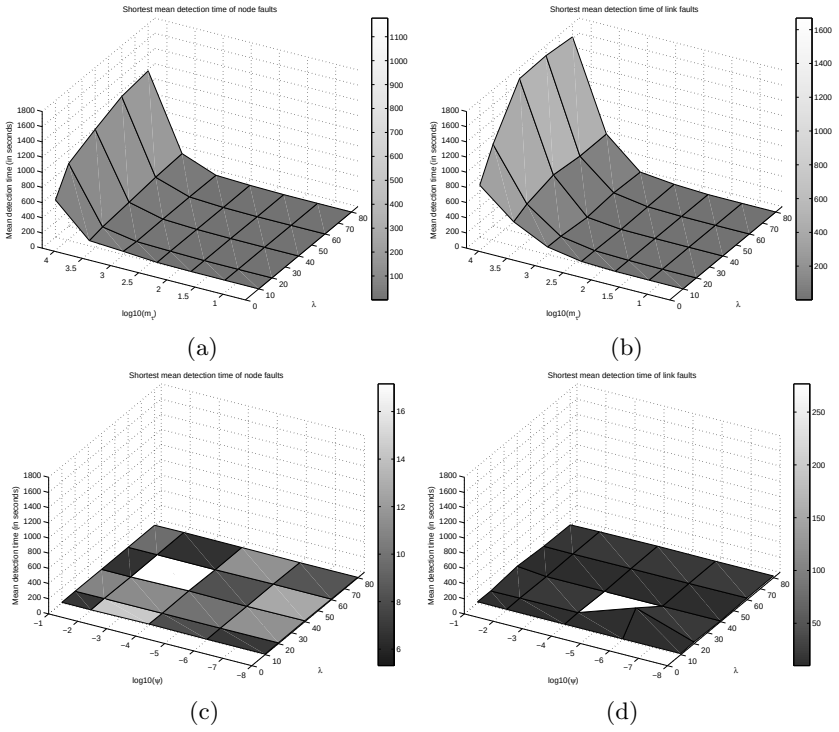
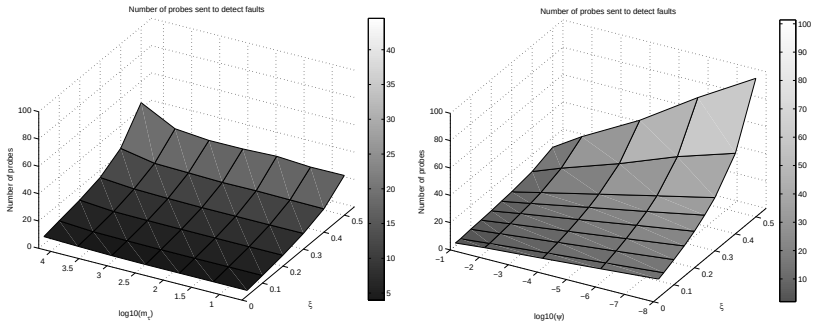


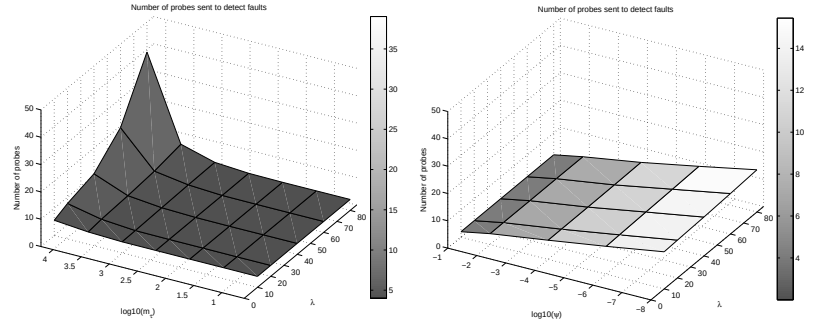
Fig. 9: Detection time with varied λ and $\xi = 0.025$. Figures 9a-9b show results obtained with varying $\tau = m_\tau f(0.8)$ and $\psi = 0.005$, whereas fig. 9c-9d show the detection time obtained with varying ψ and $m_\tau = 256$. Changes in the detection time is small up to a certain point where the interval τ is set to very large values.

fig. 8c, 8d and fig. 9c, 9d that the detection time is relatively fixed for different values on ψ.



(a) (b)

Fig. 10: Mean number of probes needed for detection of an anomaly, with varied ξ and $\lambda = 5$. Figure 10a show results obtained with varying $\tau = m_\tau f(0.8)$ and $\psi = 0.005$, whereas fig. 10b show the results obtained when varying ψ and holding $m_\tau = 256$ fixed. We observe that the number of probes needed to identify a fault is mainly dependent on ψ and the drop rate.



(a) (b)

Fig. 11: Mean number of probes needed for detection of an anomaly, with varied λ and $\xi = 0.025$. Figure 11a show results obtained with varying $\tau = m_\tau f(0.8)$ and $\psi = 0.005$, whereas fig. 11b show the results obtained when varying ψ and holding $m_\tau = 256$ fixed. The number of probes needed can be adjusted by varying the value of ψ.

In figure 10 we see that the number of probes needed to detect an anomaly clearly varies with the drop rate. Specifically, we observe that for a certain degree of packet drops, the number of probes varies with different values of ψ (fig. 10b), while remaining relatively fixed when only m_τ is varied (fig. 10a). Naturally,

for small values of ψ more probes are needed reduce the uncertainty about a potential anomaly. Further, we see in figure 11 that the number of probes needed to detect an anomaly is in general unaffected by increased number of fault events when m_τ and ψ are fixed. The peak in figure 11a occurs because when τ is set to a very large value in combination with a high failure rate (or drop rate), fewer probes are sent in normal mode and thus the parameter estimation of the Gamma distribution is based on very few observations for some nodes. Consequently, this can lead to very long probing sequences in different regions of the network. Finally, we see in figure 11b that the number of probes varies with different values of ψ, independently of the number of fault events.

4.3 Localization time and fault-localization performance

In this section we investigate the localization performance of detected faults. Specifically, we have measured the rate of correctly localized faults and the mean shortest localization time, using log files created during performed experiments with varying τ and ψ, combined with either increasing drop rate ξ or failure events λ. The localization time was measured from the time of detection of a fault event, to the time when the collaborative fault localization process was finished by the initiating node.

For different values on both τ and ψ, we see that the localization rate with respect to the number of generated fault events decreases with increasing drop rate and failure rate (fig. 12, 13). Further, we see that very high values of τ produces lower localization rates compared to the rest of the surface (fig. 12a, 12b, 13a, 13b), which matches the results from previous sections (section 4.1 and 4.2). Moreover, we see that different values on ψ have insignificant effects on the overall localization performance (fig. 12c, 12d, 13c, 13d). In all cases, we see that around 70% of the node failures and 95% of the link faults can be correctly localized for certain parameter settings (fig. 12, 13). The reduced localization rates in both the cases of increased drop rate and failure rate (fig. 12, 13) are essentially caused by increasingly ineffective communication between nodes. In combination with packet drops, no alternative routing paths and increasing number of overlapping faults, the information exchanged between nodes in the collaborative fault-localization process becomes increasingly insufficient. This means that faults are detected, but undecidable to a higher degree than when the drop rate or failure rate is low.

The mean shortest localization time is shown in figures 14 and 15. In all the experiments, we see that the localization time is less than 50 seconds for the node failures and less than 20 seconds for link failures. In most cases, node failures are localized within 20 seconds whereas link failures are localized within 5 seconds. The rather rough appearance of the resulting surfaces are caused by overlapping node and link failures, which can time the localization process. We observe from the results that the localization time for node failures tend to increase with larger values of m_τ (fig. 14a, 15a). The reason is that when m_τ is small, more nodes detect the anomaly for a common node. This triggers different localization processes with different localization time, which is at most

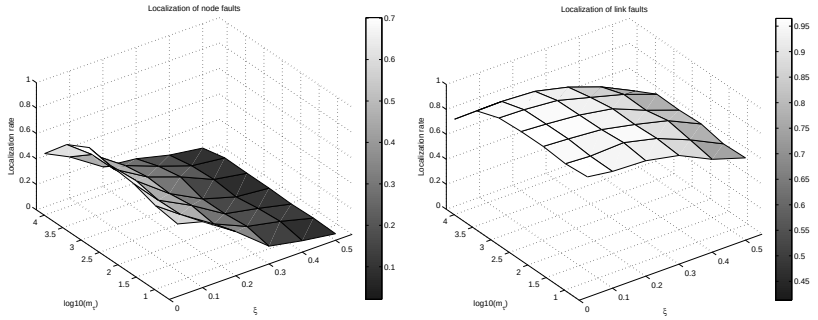


Fig. 12: Localization rate of faults with increasing ξ and fixed $\lambda = 5$. In fig. 12a-12b, we see results obtained with varying $\tau = m_\tau f(0.8)$ and $\psi = 0.005$, whereas fig. 12c-12d show experimental results obtained with varying ψ and $m_\tau = 256$. We observe that the localization rate decreases with increasing ξ.

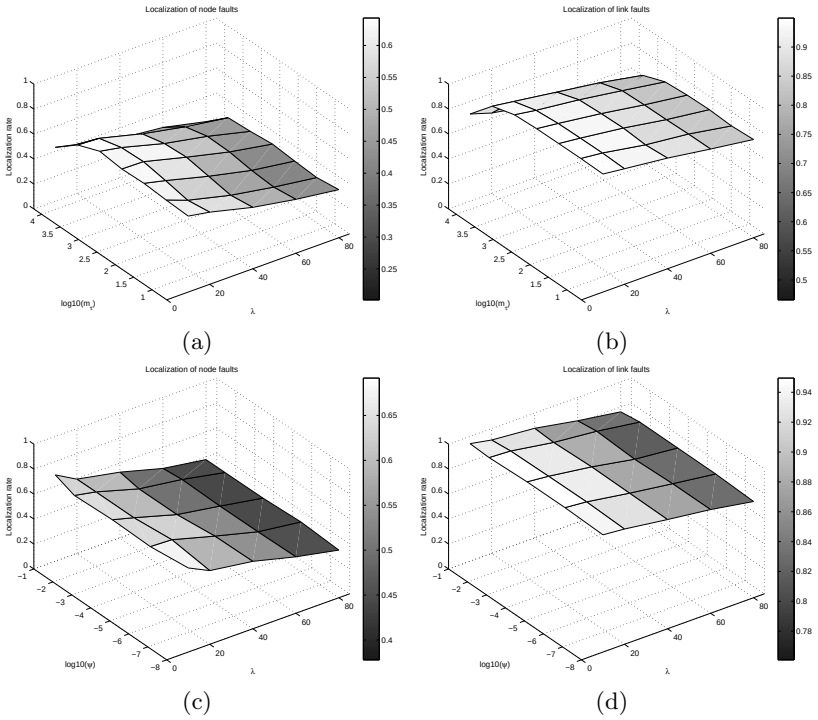


Fig. 13: Localization rate of faults with fixed $\xi = 0.025$ and varying λ. In fig. 13a and 13b, $\tau = m_\tau f(0.8)$ vary while $\psi = 0.005$ is fixed, whereas fig. 13c and 13d show results obtained with varying ψ and $m_\tau = 256$. The localization rates decreases with increasing λ.

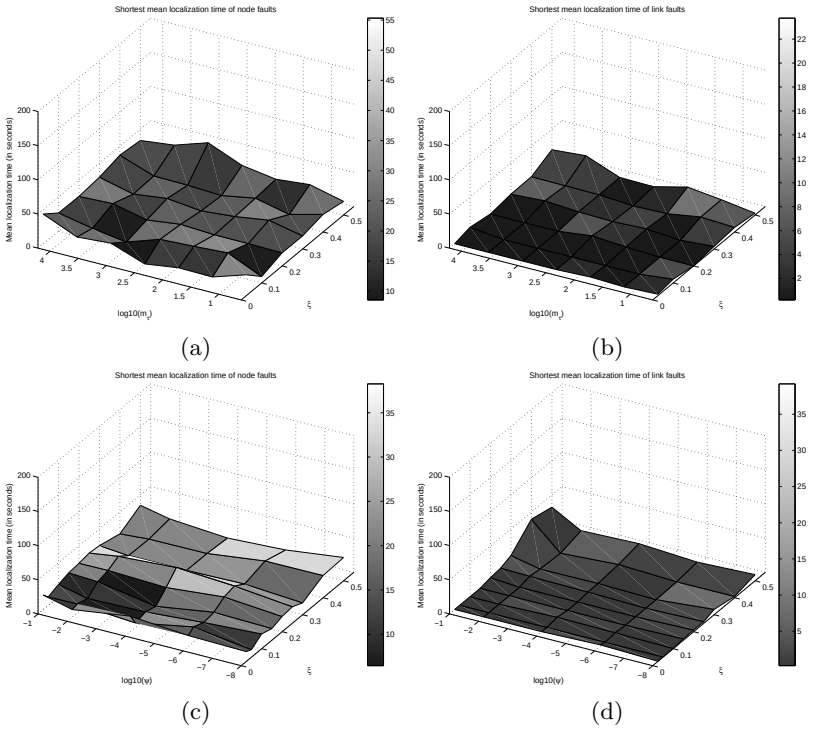


Fig. 14: Localization time with varied ξ and $\lambda = 5$. In fig. 14a-14b, we see results obtained with varying $\tau = m_\tau f(0.8)$ and $\psi = 0.005$, whereas fig. 14c-14d show results with varying ψ and $m_\tau = 256$. The localization time increases with ξ and can possibly be improved by adjusting ψ.

t_r (for undecidable faults). With a larger value of m_τ, an anomaly is most likely detected and localized by a single node, which may increase the mean shortest localization time. For the ψ, we observe in fig. 14c and 14d, a slight increase in the localization time with increasing drop rate and a fixed value of ψ, specifically for the node failures, which matches the results with increasing number of probes needed to detect (and confirm) anomalies (fig. 10b).

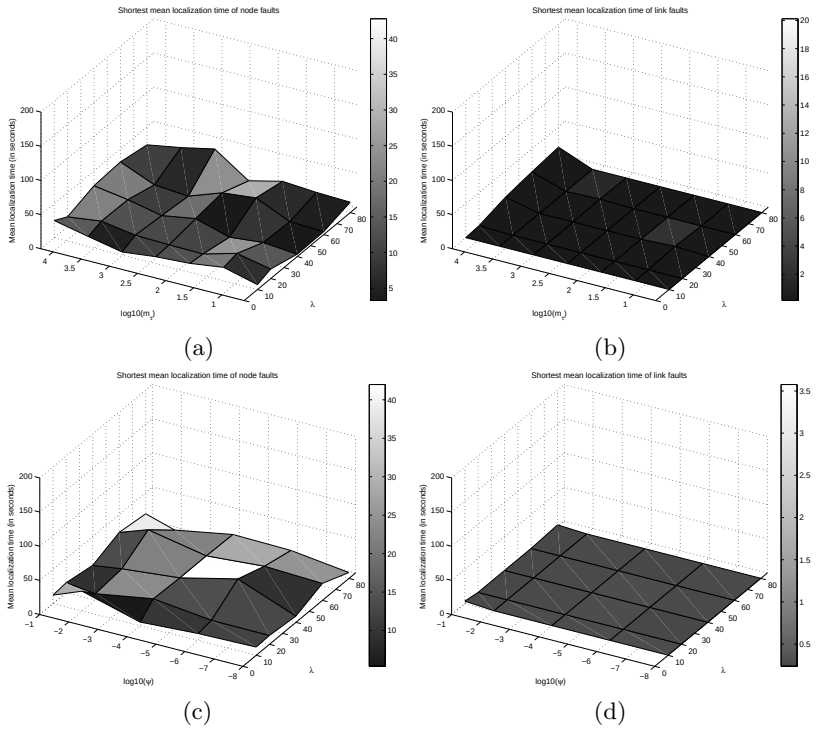


Fig. 15: Localization time with fixed $\xi = 0.025$ and varied λ. In fig. 15a-15b, we see results obtained with varying $\tau = m_\tau f(0.8)$ and $\psi = 0.005$, whereas fig. 15c-15d show results with varying ψ and $m_\tau = 256$. We see from the results that faults can be localized in less than 10 seconds.

5 Discussion

Without aiming for optimal performance, the results of performed experiments show that it is possible to detect and localize at least 90% of generated link failures and around 70% of the node failures. The results also show that traffic anomalies can be detected within 10 seconds and localized within 50 seconds. By adjusting the parameters ψ and m_τ, the performance in terms of short detection

time and localization time, false positives and correctly localized faults can be further improved.

In the current setting, it has been observed that some node failures are not resolved, simply because the routes between collaborating nodes are blocked by the faulty node in question. Introducing alternative paths for collaboration purposes could therefore improve the the algorithm performance in terms of correctly localized node failures. This, however, is out of scope of our approach to anomaly detection, and is currently up to the user (i.e. the network operator) of our algorithm to implement.

In all of the experiments, ψ was set equal over all nodes and independent to any local traffic properties. To improve both adaptivity and autonomicity of our algorithm towards a zero-configuration approach, it is desirable to let ψ in the future be set based on a cost similarly to τ. However, the latency threshold ψ should not depend on the expected link latency, as is the case for τ. The reason is that ψ in the definition of the anomaly detection model is more related to drop rate (which was also verified by the experimental results) than expected link latency - setting ψ to a multiple of the expected cost would only make the detection of a fault dependent on the actual link latency, such that reduction in the uncertainty of a detected fault on slower links would be lesser than for fast links. Instead, the ψ should co-vary with the measured drop rate in order to compensate the level of uncertainty and reduce the number of false positives. In practice, this means that we want lower ψ for high levels of traffic drops and vice versa, such that more probes are sent in order to reduce the number of false positives. It is feasible that we in the future will investigate the degree of improvement a redefinition of ψ with drop rate included might have on the performance.

In general the cost m_τ controlling the probing parameter τ is a trade-off between performance and induced link load - with a high cost, probes are sent with sparse intervals which can lead to a higher degree of undetected faults, longer detection time, and fewer false positives. However, with a small cost, probes are sent with short intervals which increases the link load and the number of false positives. Our results indicate that with a relatively large value of τ, our adaptive approach to anomaly detection performs satisfactory without inducing unnecessary link load caused by excessive probe testing. This makes our method a potential candidate for practical use, compared to other heartbeat-monitoring methods without an adaptive model.

In addition to detection of traffic anomalies caused by network component failures, we aim to extend the current model to include detection of link latency deviations. In real world networks, small deviations in the expected link latency can indicate failures in network services or equipment. This could relatively easily be used for e.g. early detection and localization of traffic anomalies caused by congestion. Thus, with an extension of our model, degeneration in traffic flow and network services can quickly be detected and resolved. For this purpose, the same model for the expected link latency described in eq. (1) can be applied in combination with a mathematical expression for the detection of the latency

deviation. In near future it is feasible to develop this kind of model extension and investigate the overall performance.

6 Conclusion

We have investigated the performance of a distributed, statistical approach to anomaly detection and fault-localization. Results of performed experiments indicate satisfactory algorithm performance - by adjusting algorithm parameters, the performance can be fine-tuned for fast responses to anomalies and accurate localizations of the root causes.

The main benefit of our statistical approach is that the time interval with which a probe is sent, and the number of probes used to reduce the uncertainty of a potential fault, are adapted to the measured link latency such that only short series of probes are needed to detect an anomaly. This way, the traffic load on the link is minimally affected by probe traffic, compared to ordinary heartbeat monitoring in which probes are sent frequently at fixed time intervals. Since probe intervals are determined autonomously for each individual link, the need for manual configuration is significantly reduced while satisfactory monitoring performance can be achieved.

As both anomaly detection and fault-localization are performed in a distributed manner, our approach should scale well with the number of network components while adapting to local conditions. Further, our method is easily used in dynamic networks with varying topology caused by so called churning (i.e. addition, removal or replacement of nodes) because of the simple initialization procedures that we have proposed. In addition, network equipment of today already fulfils most of the requirements needed to carry out described operations - the implementation of the protocols needed to run the described algorithm should therefore be relatively easy.

Future work include further investigation of parameter settings relative algorithm performance and scalability, tested on both synthetically generated networks and realistic topologies with known link latencies. Further, the algorithm will be extended to include detection of anomalies related to abnormal probe reply latencies, based on the same probabilistic model as described in this report. Finally, we aim to investigate how to further reduce the need for manual configuration towards a zero-configuration approach to anomaly detection and localization.

References

1. P. Barford, J. Kline, D. Plonka, and A. Ron. A signal analysis of network traffic anomalies. In *IMW'02*, pages 71–82, Marseille, France, 2002.
2. E. Casilari, A. Reyes-Lecuona, F. Gonzlez, A. Diaz-Estrella, and F. Sandoval. Characterization of web traffic. In *In Proc: GLOBECOM 2001*, pages 1862–1866, 2001.

3. H. K. Choi and J. O. Limb. A behavioral model of web traffic. In *ICNP '99: Proceedings of the Seventh Annual International Conference on Network Protocols*, page 327, Washington, DC, USA, 1999. IEEE Computer Society.
4. D. Ersoz, M. S. Yousif, and C. R. Das. Characterizing network traffic in a cluster-based, multi-tier data center. In *ICDCS '07: Proceedings of the 27th International Conference on Distributed Computing Systems*, page 59, Washington, DC, USA, 2007. IEEE Computer Society.
5. J. Färber. Network game traffic modelling. In *NetGames '02: Proceedings of the 1st workshop on Network and system support for games*, pages 53–57, New York, NY, USA, 2002. ACM.
6. H. Hajji. Statistical analysis of network traffic for adaptive faults detection. *IEEE Transactions on Neural Networks*, 16:1053–1063, 2005.
7. C. Heyaime-Duverge and V. K. Prabhu. Modeling action and strategy internet-games traffic. *IEEE 55th Vehicular Technology Conference*, 3:1405–1409, 2002.
8. L. Huang, X. Nguyen, M. Garofalakis, M. I. Jordan, A. Joseph, and N. Taft. In-network pca and anomaly detection. In *Advances in Neural Information Processing Systems 19*, pages 617–624. MIT Press, Cambridge, MA, 2007.
9. A. Lakhina, M. Crovella, and C. Diot. Characterization of network-wide anomalies in traffic flows. In *IMC'04*, pages 201–206, Taormina, Sicily, Italy, 2004.
10. M. V. Mahoney. Network traffic anomaly detection based on packet bytes. In *SAC'03*, Melbourne, Florida, USA, 2003.
11. M. Natu and A. S. Seti. Efficient probing techniques for fault diagnosis. *Second International Conference on Internet Monitoring and Protection (ICIMP 2007)*, pages 2085–2090, 2007.
12. M. E. J. Newman. The structure and function of complex networks. *SIAM Review*, 45:167–256, 2003.
13. S. Rangarajan, A. Dahbura, and E. Ziegler. A distributed system-level diagnosis algorithm for arbitrary network topologies. *IEEE Transactions on Computers*, 44(2):312–334, 1995.
14. I. Rish, M. Brodie, S. Ma, N. Odintsova, A. Beygelzimer, G. Grabarnik, and K. Hernandez. Adaptive diagnosis in distributed systems. *IEEE Transactions on Neural Networks*, 16:1088–1109, 2005.
15. A. Subbiah and D. M. Blough. Distributed diagnosis in dynamic fault environments. *IEEE Transactions on Parallel and Distributed Systems*, 15(5):453–467, 2004.
16. A. Varga and R. Hornig. An overview of the omnet++ simulation environment. In *Simutools '08: Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, pages 1–10, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).